CASCADE ERROR PROJECTION: A NEW LEARNING ALGORITHM

Tuan A. Duong, Allen R. Stubberud[†], Taher Daud, and Anil P. Thakoor
Center for Space Microelectronics Technology
Jet Propulsion Laboratory, California Institute of Technology
Pasadena, CA 91109
tuan@brain.jpl

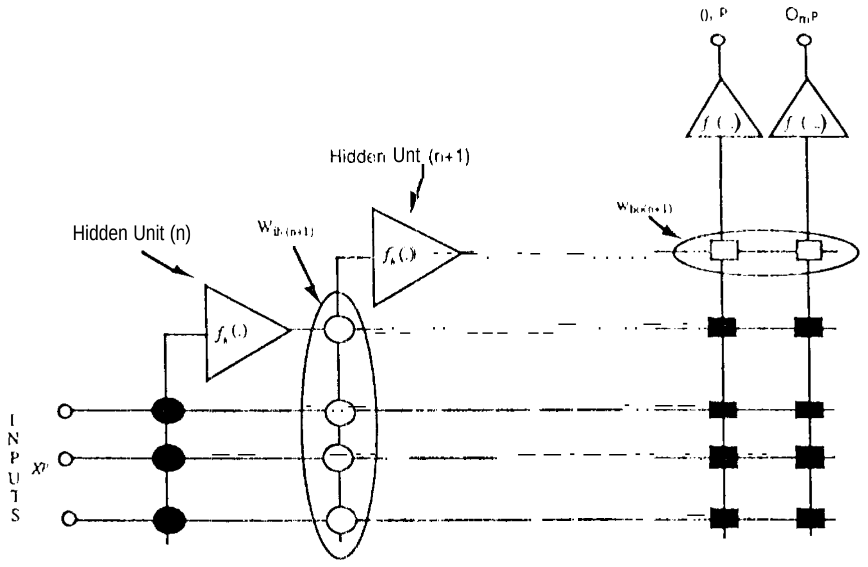


Figure 1: Schematic diagram for CEP learning with a newly added hidden unit (n+1). Blank circles and squares respectively are the weight components that are to be obtained by iterative learning and calculations. Filled circles and squares similarly are the respective weights already obtained by iterative learning and calculations.

$$f(x) = \frac{1-e^{-x}}{1+e^{-x}}$$

Other notations are defined as follows: W_{ho} denotes the weight vector between newly added hidden unit $n+1$ and the output o; W_{ih} is the weight vector between input units (original input and previous hidden units) and a newly added hidden unit; $\varepsilon_o^p = t_o^p - o_o^p(n)$ denotes the error for an output index o and training pattern p between target t and the actual output *o(n)* with n hidden units in the network; $f'^p_o(n)$ denotes the derivative of the output transfer function with respect to its input for an output index o and the training pattern p; $f_h^p(n+1)$ denotes the transfer function of hidden unit $n+1$ for a training pattern p; and X^p denotes the input vector with number of patterns p.

The energy function is defined as:

$$E(i) = \sum_{p=1}^{P} E^p(i) = \sum_{p=1}^{P}\sum_{o=1}^{m}(t_o^p - o_o^p(i))^2 = \sum_{p=1}^{P}\sum_{o=1}^{m}(\varepsilon_o^p)^2$$

The difference of energy between the network with n hidden units and that with $n+1$ hidden units is obtained as:

$$\Delta E = E(n) - E(n+1) = \sum_{o=1}^{m}\{-w_{ho}^2\sum_{p=1}^{P}[f'^p_o f_h^p(n+1)]^2 + 2w_{ho}\sum_{p=1}^{P}[\varepsilon_o^p f'^p_o f_h^p(n+1)]\}$$

where $w_{ho} f_h^p(n+1)$ is small. This assumption is needed for nonlinear transformation function only. One can derive the maximum energy reduction of the circuit from n hidden units to $n+1$ hidden units with respect to w_{ho} as:

$$\max\{(\Delta E)_{w_{ho}}\} = \sum_{p=1}^{P}\sum_{o=1}^{m}\{\varepsilon_o^p f'^p_o f_h^p(n+1)\} \quad \text{provided,} \quad w_{ho} = \frac{\sum_{p=1}^{P}\varepsilon_o^p f'^p_o f_h^p(n+1)}{\sum_{p=1}^{P}[f'^p_o f_h^p(n+1)]^2} \qquad (1)$$

Let us define the vector:

$$\Gamma = \begin{bmatrix} \frac{1}{m}\sum_{o=1}^{m} f'^{1}_{o}\{t^{1}_{o} - o^{1}_{o}\} \\ \cdots\cdots \\ \cdots\cdots \\ \cdots\cdots \\ \frac{1}{m}\sum_{o=1}^{m} f'^{P}_{o}\{t^{P}_{o} - o^{P}_{o}\} \end{bmatrix}$$

Then, $\Gamma \in [-1,1]^P$, and

$$F_h(n+1) = \begin{bmatrix} f^1_h(n+1) \\ \cdots\cdots \\ \cdots\cdots \\ \cdots\cdots \\ f^P_h(n+1) \end{bmatrix}$$

We can, therefore, rewrite equation (1) using matrix notation as follows:

$$\text{Max } \Delta E = m\Gamma^T F_h(n+1) \tag{2}$$

From (1) and (2), the energy reduction is dependent on a correlation between Γ and F_h (n + 1) The idea now is to modify $F_h(n+1)$ by training to obtain as good a match with Γ as possible before the next hidden unit is added. To obtain this match, any of the techniques, e.g. perception learning with gradient descent, maximum correlation, covariance with gradient ascent conjugate gradient, or Newton's second order method may be used, Unlike cascade correlation (CC) technique's; however, only the weights W_{ih} are to be learned here. That done, f_h is known, and hence w_{ho} can be calculated from Eqn. (1). Therefore, the learning network performance really depends on the learning technique chosen for matching the error surfaces Γ and $F_h(n+1)$. If we let $f'^{P}_{o}(n) = 1$; then Eqn. (2) can be rewritten as:

$$\Delta E = \sum_{p=1}^{P}\{f^P_h(n+1)\frac{1}{m}\sum_{o=1}^{m}(t^P_o - o^P_o(n))\} \tag{3}$$

~'bus, equation (3) is a special case of the general formulation of Eqn. (2), When one maximizes ΔE in Eqn. (3), then one obtains the ('C learning algorithm,

IV Cascade Error Projection Learning Algorithm and Simulation

a) ***Learning approach:***

From Eqn. (2), a new energy function is defined as:

$$\Phi(n+1) = \sum_{p=1}^{P}\{f^P_h(n+1) - \frac{1}{m}\sum_{o=1}^{m}(t^P_o - o^P_o)f'^P_o\}^2$$

'I'he weight updates between the inputs (including the weights by expanded inputs) and the newly added hidden unit arc calculated as follows:

$$\Delta w'^{P}_{ih}(n+1) = -\eta\frac{\partial\Phi(n+1)}{\partial w'^{P}_{ih}(n+1)}$$

where η is the learning rate, and the updated weight value for the synapse between the hidden unit h ant] the output unit o, [$w_{ho}(n+1)$], can be calculated from Eqn (I)

b) ***Simulation***

1) *Parity Problems* Using (his technique, we have solved, 5- to 8-bit parity problems with (1) no limited weight quantization (floating point 32-bit for single precision and 64-bit for double precision); and, (2) the limited weight quantization from 3 to 6 bits for the hardware,

2) *Conversion technique (round-off technique)* In continuous weight space, the weight quantization can be considered as infinite. However, in hardware, weight quantization is always finite and limited. Therefore, it is necessary to convert the weight updates Aw to a finite weight quantization Aw*, It can be shown that learning can be done with limited weight quantization as long as the difference between Aw and Aw* is viewed as equivalent independent white noise (round-off conversion technique) and the stepsize which is used to convert from Aw to Aw* must not be fixed. The dynamical stepsize can be roughly estimated as follows[9]:

$$stepsize(n+1) \propto \tilde{E}(n) \qquad (4)$$

where $\tilde{E}(n)$ is the energy level with limited weight resolution corresponding to $E(n)$ in the continuous weight space. The expression in (4) is a critical step in estimating, the dynamical stepsize which is dependent on the previous energy of the network. In other words, the expression can be written as:

$$stepsize(n+1) = \alpha\tilde{E}(n)$$

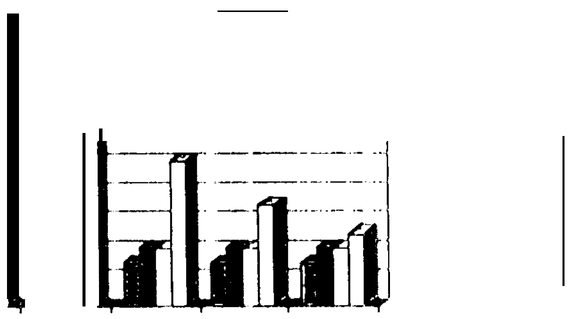
where α is a constant, and its value, as is the case with η, can be obtained for each application through trial and error (e.g. see Table 1). The weight update Δw is converted into the equivalent available weight quantization Aw*. The conversion can be summarized as follows:

$$\Delta w^{*}_{jh}(n) = \begin{cases} stepsize(n) * \mathrm{int}\left(\frac{\Delta w_{jh}}{stepsize(n)} + 0.5\right) \text{ if } \left(\frac{w_{jh}(n)}{stepsize(n)} + \mathrm{int}\left(\frac{\Delta w_{jh}(n)}{stepsize(n)} + 0.5\right)\right) \leq 2^{B} \text{ and } \Delta w_{jh}(n) > 0 \\ stepsize(n) * \mathrm{int}\left(\frac{\Delta w_{jh}(n)}{\quad} - 0.5\right) \text{ if } \left(\frac{w_{jh}(n)}{\quad} + \mathrm{int}\left(\frac{\Delta w_{jh}(n)}{\quad} - 0.5\right)\right) \leq -2^{B} \text{ and } \Delta w_{jh} \end{cases}$$

	5-bit parity	6-bit parity	7-bit parity	8-bit parity
Floating-point Weight value	η_0 = 1.0 α = N/A	η_0 = 1.0 α = N/A	η_0 = 0.4 α = N/A	η_0 = 0.4 α = N/A
3-bit Weights	η_0 = 1.0; α = .0024810	η_0 = 1.0; α = .016597	η_0 = 1.0; α = .008766	η_0 = 1.0; α = .004101

4) *Simulation results* As noted earlier, 5- ,6-, 7-, and 8-bit parity problems are solved, each with different synaptic weight resolution. The results of higher and lower resolution are compared (Fig. 2) to show the robustness of the algorithm for hardware implementation. The values of input and output highs are 0.8, and that of lows arc -0.8. The neuron transformation function is a hyperbolic tangent. Zero values are used for initial weights of each newly added hidden unit; therefore, it is not needed to conduct extra runs for each problem. A 100-epoch iterations learning is applied for each added hidden unit for the weights between inputs and the current hidden unit only.

As a comparison for



VII References:

[1] T. A. Duong, 1'. Brown, M. Tran, H. Langenbacher, and 1. Daud, "Analog VLSI neural network building block chips for hardware-in-the-loop learning," *J'roe, IEEE/INNS Int'l. Joint Conf. on Neural Networks,* Beijing, China, Nov. *3-6, 1992.*

[2] T. A. Duong et. al, "Low Power Analog

[3]

[4]

[5]

[6]

[7]

[8]

[9]